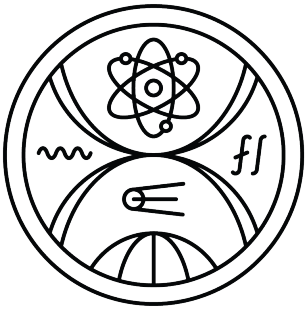
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

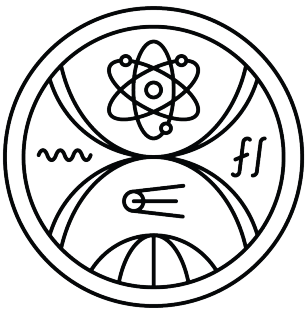


VÝVOJ SOFTVÉRU VO VIRTUÁLNEJ ALEBO ROZŠÍRENEJ REALITE

Diplomová práca

2025 Bc. Jozef Kolek

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



VÝVOJ SOFTVÉRU VO VIRTUÁLNEJ ALEBO ROZŠÍRENEJ REALITE

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. Ing. Ivan Polášek, PhD.
Konzultant:	Ing. Juraj Vincúr, PhD

Bratislava, 2025 Bc. Jozef Kolek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jozef Kolek
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vývoj softvéru vo virtuálnej alebo rozšírenej realite
Software development in virtual or extended reality

Anotácia: Programátori pri vývoji softvéru najčastejšie pracujú so zdrojovým kódom v rámci vývojárskeho prostredia pomocou klávesnice a myši, a tento kód im je zobrazovaný prostredníctvom malého počtu 2D monitorov. Takýto tradičný prístup ignoruje možnosti, ktoré nám poskytujú novovzniknuté technológie
pre zobrazovanie virtuálnej reality ako napr. Oculus Quest alebo obohatenej reality ako napr. XREAL Light.
Veríme, že vhodná migrácia jednotlivých funkcionalít vývojárskeho prostredia do sveta virtuálnej alebo obohatenej reality zvýši produktivitu programátorov, zlepší ich používateľský zážitok a skráti čas potrebný na oboznámenie programátora s neznámym softvérovým systémom.

Cieľ: Analyzujte existujúce podporné nástroje a prostredia určené na vývoj softvéru. Zamerajte sa najmä na 3D riešenia. Analyzujte už identifikované problémy, nedostatky a návrhy vylepšení existujúcich prístupov.
Navrhnite a implementujte vlastný prístup, v rámci ktorého aplikujete technológie
virtuálnej alebo obohatenej reality v kontexte vývoja softvéru.
Vyhodnoťte prínos vášho riešenia v porovnaní s tradičnými prístupmi.

Literatúra:

Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonapalawong, Tianye Zhang, Mingliang Xu and Wei Chen. 2020. A systematic literature review of modern software visualization. J. Vis., 23(4), 539–558.

Anthony Elliott, Brian Peiris, and Chris Parnin. 2015. Virtual reality in software engineering: affordances, applications, and challenges. In Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15), Vol. 2. IEEE Press, Piscataway, NJ, USA, 547-550.

F. Fittkau, A. Krause and W. Hasselbring, "Exploring software cities in virtual reality," 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), Bremen, 2015, pp. 130-134.

Vedúci: doc. Ing. Ivan Polášek, PhD.
Konzultant: Ing. Juraj Vincúr, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 10.11.2023

Dátum schválenia: 11.11.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Vyhlasujem, že som diplomovu prácu vypracoval samostatne na základe svojich vedomostí a skúseností a s použitím uvedenej odbornej literatúry.

. . . . . .

Bratislava, 2025 Bc. Jozef Kolek

Poďakovanie

Chcem poďakovať môjmu ... za jeho cenné rady a usmernenia, ktoré mi pomohli napísať túto diplomovú prácu.

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Terminológia

Pojmy

- - .

Skratky

- **VR** - virtuálna realita.

Kapitola 1

Úvod

Virtuálna realita (VR) predstavuje revolučný prístup k interakcii s digitálnymi technológiami, ktorý otvára nové možnosti aj v oblasti softvérového vývoja. Napriek jej potenciálu však tradičné metódy programovania v tomto prostredí narážajú na obmedzenia, ako sú neefektívne prechody medzi vizualizáciou a editáciou, či nutnosť opakovaného skladania VR headsetu. Táto diplomová práca sa zaoberá návrhom prostredia, ktoré umožní integrovanú prácu so softvérovými modelmi vo VR, čím eliminuje tieto bariéry a ponúka intuitívne a efektívne riešenie.

Kľúčovou súčasťou tejto práce je implementácia prepojenia UML diagramov – konkrétne Class a Activity diagramov – s vývojovým procesom, pričom sa zameriava na ich konverziu priamo do zdrojového kódu. Tento prístup nielenže zjednodušuje vývoj softvéru, ale aj minimalizuje potrebu manuálneho programovania, čím šetrí čas a znižuje riziko chýb. Cieľom je vytvoriť platformu, ktorá poskytne na jednom mieste nástroje na vizualizáciu, editáciu a automatizovanú konverziu softvérových návrhov priamo vo VR prostredí. Tento koncept spája teoretické poznatky s praktickými aplikáciami a ponúka nový spôsob, ako využiť potenciál virtuálnej reality v softvérovom inžinierstve.

Kapitola 2

Analyza

Vývoj softvéru má už skoro 150-ročnú históriu. V roku 1883 Ada Lovelace napísala algoritmus na výpočet Bernouliho schémy pre Charles Babbagov mechanický počítač. Prvý kód vznikol tak vďaka pohľadu, že čísla s ktorými počítač pracuje môžu reprezentovať aj niečo iné než len procesy v počítači. Po napísaní prvého kódu sa programovanie zmenilo výrazným spôsobom. V roku 1949 vznikol assembler inak nazývaným ako jazyk adries. Vďaka symbolovej reprezentácii zdrojového kódu sa zjednodušilo zadávanie kódu do stroja. Z nízko úrovňového Assembleru sa následne postupne vyvíjali dodnes najpoužívanejšie jazyky ako je C, Java či fenomén dnešnej doby Python.[12]

Prvý počítač vznikol v roku 1943 v Anglicku, ktorého fungovanie opísal Alan Turing vo svojej teórii pod názvom Touringov stroj. Okrem kódu sa postupne vyvíjali aj technológie na ktorých bol kód vyvíjaný a testovaný. Prvé kódy nám umožnili vznik počítača menom Eniac v roku 1946. Meral 30 metrov a spotrebu elektriny mal takú vysokú, žeby stačila na osvetlenie jedného menšieho mesta. Eniac bol najrýchlejší počítač najbližších 6 rokov, kým ho nenahradili rýchlejšie stroje pracujúce v dvojkovej sústave. Ďalší mohutný vývoj počítačov sme zaznamenali v niekoľkých vlnách. Vykryštalizovali sa nám do podoby, ktorú poznáme dnes. V dnešnej dobe máme totižto už počítače, ktoré majú násobne menšie rozmery a samozrejme aj menšiu spotrebu. Okrem počítačov máme možnosť pracovať aj s ich zmenšenou formou ako sú mobilné telefóny, notebooky, tablety či iné technológie.[8]

Vďaka vývoju programovacích jazykov a počítačov sa vyvíjal aj spôsob písania kódu a jeho vizualizácie. V minulosti sa zadávali binárne inštrukcie cez prepínače alebo dierne štítky. Kontrola kódu prebiehala častokrát manuálnym odkrokovaním na papieri alebo sledovaním výsledkov simulácií na stroji. Príchod assemblera umožnil textové písanie inštrukcií do stroja a vďaka prekladu adreies pomohol dané funkcie lahšie vykonávať. Dnes už existuje veľa možností ako vyvíjať softvér. Štandardným spôsobom je vývoj softvéru na počítači. Kód je písaný v textovom editore, z ktorého nám je pomocou kompilátora alebo interpretera generovaná aplikácia. Tento spôsob programovania prináša

so sebou množstvo textu písaného pomocou klávesnice. Aj keď prišli nejaké inovácie ako návrhy na dopĺňanie kódu alebo zadávanie kódu hlasom, stále sa pri vývoji vyžaduje väčšie množstvo ručne písaného kódu. Ďalším z problémov je vizualizácia softvéru. Ľudia nemajú veľa možností, ako zobraziť architektúru softvéru, ktorá by im uľahčila programovanie.

Jednou zo zaujímavých, zatiaľ nie veľmi preskúmaných oblastí na tvorbu softvéru je virtuálna realita. V nasledujúcich kapitolách si preto rozoberieme pojmy ako virtuálna alebo rozšírená realita (argumented reality), programovanie v nej či spôsob vizualizácie softvéru.

2.1 Realita

Veda definuje realitu ako súbor všetkých fyzikálnych javov, ktoré môžeme pozorovať a merať. Tento prístup sa opiera o empirické dôkazy a experimenty na overenie hypotéz o svete okolo nás. Realita je to svet okolo nás, tak ako prirodzene vnímame. [10]

2.2 Rozšírená realita (Augmented reality)

Rozšírená realita (AR) je technológia, ktorá kombinuje prvky virtuálneho sveta s naším skutočným prostredím, čím vytvára interaktívny zážitok. Na rozdiel od virtuálnej reality (VR), ktorá úplne nahrádza reálne prostredie digitálnym, AR pridáva digitálne prvky do reálneho sveta, ktoré môžu byť vizuálne, zvukové alebo iné senzorické informácie. AR umožňuje užívateľom vidieť a interagovať s digitálnymi objektmi, ktoré sú umiestnené do reálneho prostredia. Tieto objekty môžu reagovať na pohyby užívateľa alebo na zmeny v prostredí. AR môže byť zobrazovaná prostredníctvom smartfónov, tabletov, AR okuliarov alebo iných zariadení, ktoré využívajú kamery a senzory na mapovanie reálneho prostredia a umiestňovanie digitálnych objektov. AR sa využíva v mnohých oblastiach, ako sú hry, vzdelávanie, zdravotníctvo, marketing a priemysel. Napríklad, v zdravotníctve môže AR pomôcť chirurgom pri operáciách tým, že zobrazuje dôležité informácie priamo na pacientovi. Rozšírená realita má potenciál výrazne zmeniť spôsob, akým interagujeme s digitálnym obsahom a reálnym svetom, a otvára nové možnosti pre inovácie v rôznych oblastiach.

2.3 Zmiešaná realita

V poslednom desaťročí sa zmiešaná realita stala čoraz atraktívnejšou. Vďaka schopnosti spojiť reálny svet s digitálnymi informáciami vytvára nový vylepšený elektronický svet. Jej vznik môžeme hľadať v 60. - 70. rokoch minulého storočia kedy v roku

1968 Sutherland navrhol a dokončil svoj prvý náhlavný displej. Jej neskorší vývoj síce priniesol možnosť interakcie dokonca v 3D priestore no bol brzdený nedostatočným množstvom zariadení na zobrazovanie a vývoj aplikácií v zmiešanej realite.[12] Dnes zmiešaná realita prináša zaujímavé možnosti ako vizualizovať vývoj softvéru. Vďaka množstvu kamier a senzorov je náhlavný displej schopný sledovať náš pohľad, počúvať náš hlas či sledovať pohyb našich rúk. Tieto veci nám umožňujú v nekonečnom priestore manipulovať s objektami, otvárať ich a zatvárať. Okrem výhod trápi zmiešanú realitu obmedzený výkon zariadenia či výdrž batérie ale takisto aj problém identifikovania polohy objektov. To spôsobuje problémy pri ovládaní. V neposlednom rade sa v dnešnej dobe rieši správna kombinácia zobrazenia objektu v reálnom svete. Aby sa zabránilo nehodám v správny čas sa vykreslia predmety z reálneho sveta miesto virtuálnych objektov. Táto funkcia má dnes ešte veľké nedostatky a bude trvať ešte niekoľko rokov kým sa stane zo zmiešanej reality bežný nástroj akým sú dnes počítače. Tak či tak zmiešaná realita predstavuje veľmi zaujímavé prostredie pre vývoj a vizualizáciu softvéru. V ďalších kapitolách si jeden z takýchto prístupov podrobnejšie rozoberieme.

Obr. 2.1: Príklad zmiešanej reality
Zroj: Obrázok generovaný Microsoft Copilot

2.4 Obohatená Virtualita

Obohatená virtualita (Augmented Virtuality - AV) je technológia, ktorá kombinuje prvky virtuálnej reality (VR) a rozšírenej reality (AR) tým, že integruje reálne objekty do prevažne virtuálneho prostredia. Na rozdiel od rozšírenej reality, kde digitálne prvky dopĺňajú reálny svet, AV prináša reálne prvky do digitálneho sveta. AV ponúka úroveň ponorenia a interakcie, ktorá presahuje tradičnú rozšírenú realitu, umožňujúc užívateľom manipulovať a zapájať sa do integrovaných reálnych prvkov. Pomocou AV môžeme

vytvoriť realistické tréningové prostredie pre letcov, chirurgov, vojakov a arhcitektov. AR je aktraktívna tiež pre tvorcov hier.

Obr. 2.2: Rozdiel medzi jednotlivými realitami
Zdroj: https://xr4all.eu/xr/

2.5 Virtuálna realita

Virtuálna realita (VR) je technológia, ktorá umožňuje užívateľom vstúpiť do počítačom generovaného prostredia, ktoré simuluje fyzickú prítomnosť v reálnom alebo imaginárnom svete. Prostredníctvom špeciálnych zariadení, ako sú VR okuliare a rukavice, môžu užívatelia interagovať s týmto prostredím v reálnom čase. VR sa stáva čoraz populárnejšou nielen v zábavnom priemysle, ale aj v rôznych profesionálnych oblastiach, vrátane medicíny, vzdelávania a architektúry. Jednou z najvýznamnejších aplikácií VR je vizualizácia softvéru. Táto technológia umožňuje vývojárom a dizajnérom vytvárať a testovať softvérové aplikácie v trojrozmernom prostredí, čo prináša niekoľko výhod. Po prvé, VR poskytuje realistickejší a intuitívnejší spôsob, ako si predstaviť a manipulovať s komplexnými dátovými štruktúrami a algoritmami. Napríklad, vývojári môžu vizualizovať tok dát v reálnom čase, čo im umožňuje lepšie pochopiť a optimalizovať výkon ich aplikácií. Po druhé, VR umožňuje tímom spolupracovať v zdieľanom virtuálnom priestore, čo je obzvlášť užitočné pre distribuované tímy pracujúce na veľkých projektoch. V takomto prostredí môžu členovia tímu diskutovať o dizajnových rozhodnutiach, identifikovať problémy a testovať rôzne riešenia bez nutnosti fyzickej prítomnosti na jednom mieste. To nielen zvyšuje efektivitu, ale aj znižuje náklady spojené s cestovaním a logistikou. Napokon, VR môže byť využitá aj na školenie a vzdelávanie softvérových inžinierov. Virtuálne prostredie umožňuje simulovať rôzne scenáre a problémy, s ktorými sa môžu vývojári stretnúť v reálnom svete, čím sa zvyšuje ich pripravenosť a schopnosť riešiť komplexné úlohy. Taktiež poskytuje bezpečný priestor na experimentovanie a učenie sa bez rizika poškodenia reálnych systémov. V závere môžeme povedať, že virtuálna realita prináša revolučné možnosti do oblasti vizualizácie softvéru. Jej schopnosť vytvárať interaktívne a pohlcujúce prostredia otvára nové cesty

pre vývoj, spoluprácu a vzdelávanie, čím prispieva k rýchlejšiemu a efektívnejšiemu vývoju softvérových riešení.

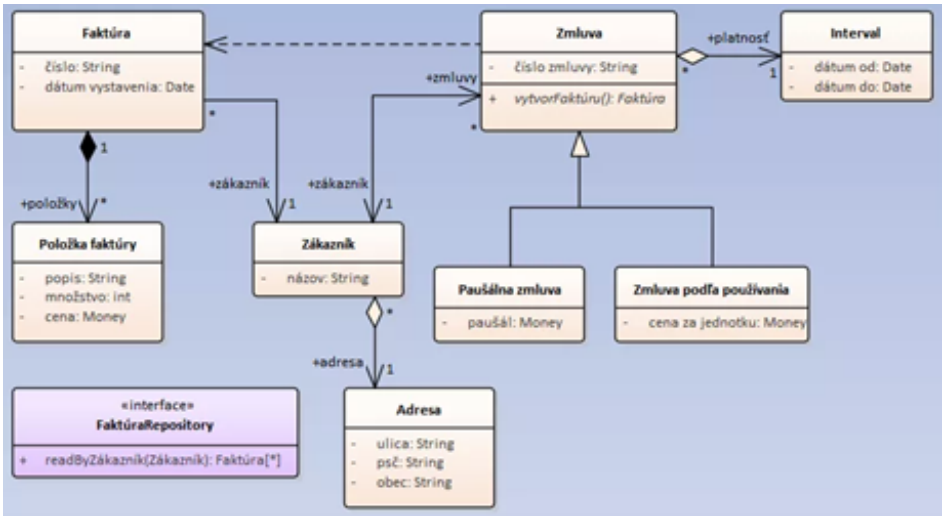
2.6 UML

V softvérovom inžinierstve vznikajú často krát problémy pri špecifikácii a analýze požiadaviek od zákazníka. UML jazyk prináša so sebou niekoľko zaujímavých objektovo orientovaných techník, ktoré pomáhajú premietnuť problémy reálneho sveta do sveta počítačovej logiky. UML predstavuje súbor diagramov, ktorými vieme špecifikovať rôzne požiadavky alebo návrh pre softvérový systém. Poskytuje nám súbor notácií určených na vizualizáciu a tvorbu artefaktov. Jedným z mnohých využití je modelovanie obchodných procesov, definovanie interakcií medzi systémovými komponentami alebo návrh softvérových architektúr. Nakoľko UML jazyk obsahuje v sebe veľa spôsobov na vizualizáciu požiadavkiek a vlastností softvéru, nevyhli sa mu problémy s tým spojené. Notácie častokrát obsahujú formálne alebo poloformálne definície, vďaka ktorým dochádza k nejasnostiam pri pochopení modelu. Ďalším z problémov je existencia viacerých UML zápisov, ako dané požiadavky vizualizovať. Vďaka tomu môže vzniknúť nekonzistentný a nejednoznačný model. V neposlednom rade môže dôjsť k rôznym interpretáciám na strane príjemcu, ktorý daný model buď pochopí ináč alebo mu nebude dostatočne jasný. Ako riešenie sa ponúka dodatočne doplniť alebo pridať definície pre jednotlivé diagramy, aby sa tak vyhlo opakovaným problémom. Napriek problémom, ktoré UML má, nám ponúka širokú škálu metód, ktoré sú nielen vhodným pomocníkom pri návrhu alebo analýze požiadaviek ale predstavujú aj zaujímavú formu pre vývoj softvéru. V neskorších kapitolách si preto podrobnejšie rozoberieme rôzne prístupy ako vizualizovať softvér pomocou UML a najmä ako vyvíjať softvér vo virtuálnej realite. Predtým však v krátkosti ukážeme niekoľko UML metód, vhodných pre túto vizualizáciu.[4]

2.7 Diagram tried

Diagram tried je statický diagram. Okrem vizualizácie rôznych aspektov systému ho vieme použiť pri zostavovaní spustiteľného kódu. Diagram je všeobecne zložený z množiny tried a vzťahov medzi nimi.[13] Diagram je všeobecne zložený z množiny tried a vzťahov medzi nimi. Každá trieda obsahuje:

- **Meno:** typicky je písané v hornej časti triedy
- **Atribúty:** sú dátovými členmi triedy, reprezentujú jej vlastnosti
- **Operácie:** reprezentujú funkčné správanie triedy

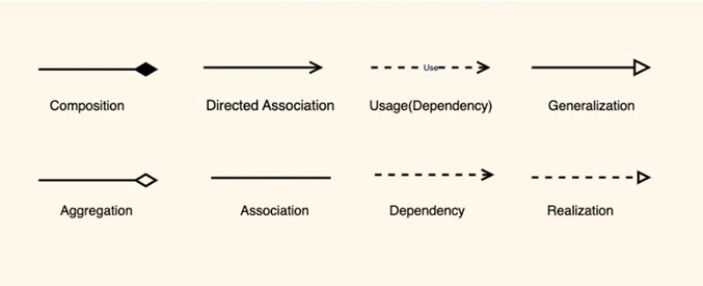


Obr. 2.3: Príklad UML diagramu modelujúci evidenciu zmlúv pre zákazníka
Zdroj: https://dddcommunity.sk/vztahy-v-objektovom-svete/

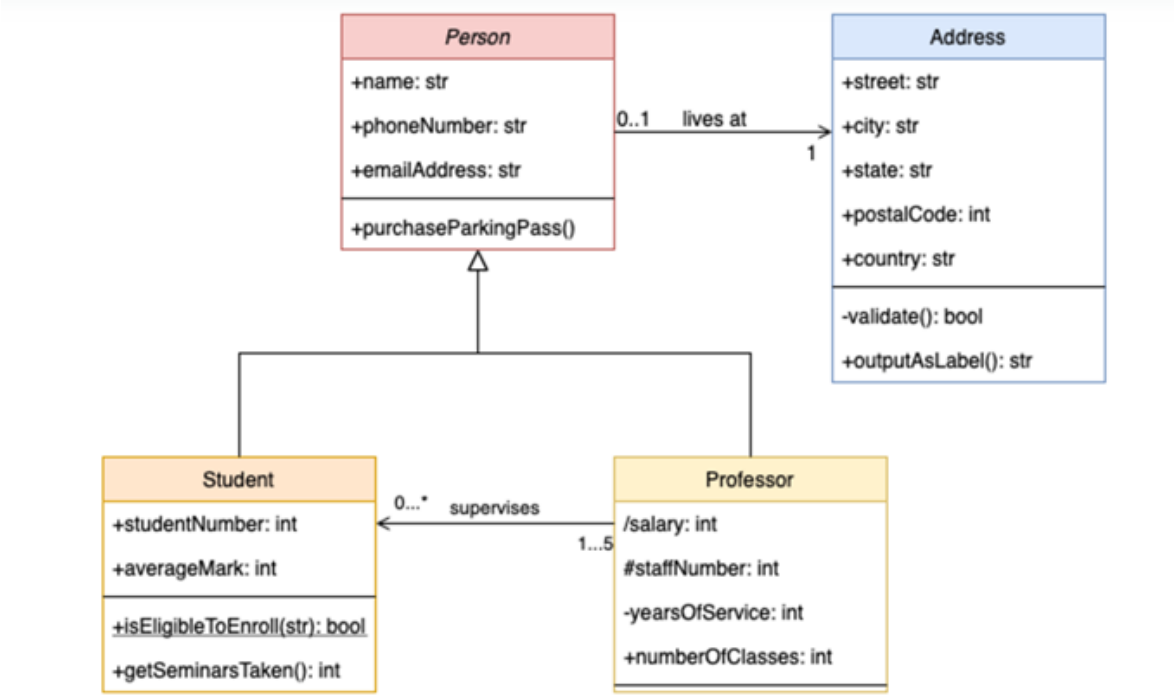
V diagrame tried identifikujeme spolu 8 vzťahov. Sú nimi:

- **kompozícia:** je formou agregácie, kde časť triedy nemôže fungovať samostatne bez druhej
- **priama asociácia:** reprezentuje jednosmerný vzťah medzi triedami
- **(usage) závislosť:** jedna trieda nie je schopná vykonať operáciu bez funkcionalít tej druhej
- **generalizácia:** alebo aj dedičnosť je keď jedna trieda pri výstavbe seba zdedí metódy aj atribúty tej druhej
- **agregácia:** symbolizuje silnú formu asociácie kde jedna trieda je založená na základe druhej triedy
- **asociácia:** reprezentuje obojstranný vzťah medzi inštanciami tried
- **závislosť:** keď sa jedna trieda spolieha na tú druhú. Vzťah nie je však tak silný ako asociácia či generalizácia
- **realizácia:** implementuje funkcie iného interfejsu

Diagramom tried vieme z kódu vhodne vizualizovať triedy ich definované metódy, atribúty a aj interfejsy a namespacy. Okrem tohto je dôležité špecifikovať potrebné vzťahy medzi triedami ako sú dedičnosť, či použitie inštancie triedy v inej triede, poprípade využitie jej vlastností a funkcionalít. Zvyšné funkčné časti kódu vieme doplniť pomocou sekvenčného diagramu alebo diagramu aktivít. Diagram tried predstavuje teda vhodný nástroj na vizualizáciu softvéru.[1]



Obr. 2.4: Vzťahy v diagrame tried
Zdroj: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/



Obr. 2.5: Diagram tried príklad
Zdroj: https://medium.com/codex/concepts-you-need-to-know-about-object-oriented-programming-d168faa3bbc1

2.8 Sekvenčný diagram

Sekvenčné diagramy sú interakčné diagramy, ktoré znázorňujú vykonávanie jednotlivých operácií alebo ich špecifických prípadov. Diagram sa pri nich sústreďuje aj na poradie, v ktorom sa vykonávajú. Vizuálne znázorňuje diagram proces medzi objektami alebo ich inštanciami v čase, pričom poradie objektov je organizované vodorovne a postupnosť akcií v čase t je organizovaná vertikálne.

Sekvenčný diagram sa skladá z dvoch typov objektov. Sú nimi:

- **účastník:** reprezentuje nejakú čiastočnú rolu entity
- **línia života:** predstavuje individuálneho účastníka procesu

- **aktivácia:** reprezentuje ju obdĺžnik ktorý znázorňuje svojou výškou čas za ktorý daný účastník vykonáva danú operáciu

Medzi základné operácie v diagrame patria:

- **poslanie správy:** správa definuje komunikáciu medzi objektami. Jej poslaním sa vyvolá proces u druhého objektu
- **spätná správa:** objekt, ktorý na žiadosť druhého objektu proces vykonal pošle spätne informácie o vykonaní druhému objektu

Ďalšími zaujímavimi typmi operácií (vzťahov) sú aj poslanie správy samému sebe, rekurzívne poslanie správy, či správy o vymazanie. V neposlednom rade máme v sekvenčnom diagrame fragmenty, ktoré nám pomáhajú vhodne izolovať špeciálne prípady procesov alebo opakovane niektoré procesy vykonávať. Spomenieme len niekoľko známejších fragmentov:

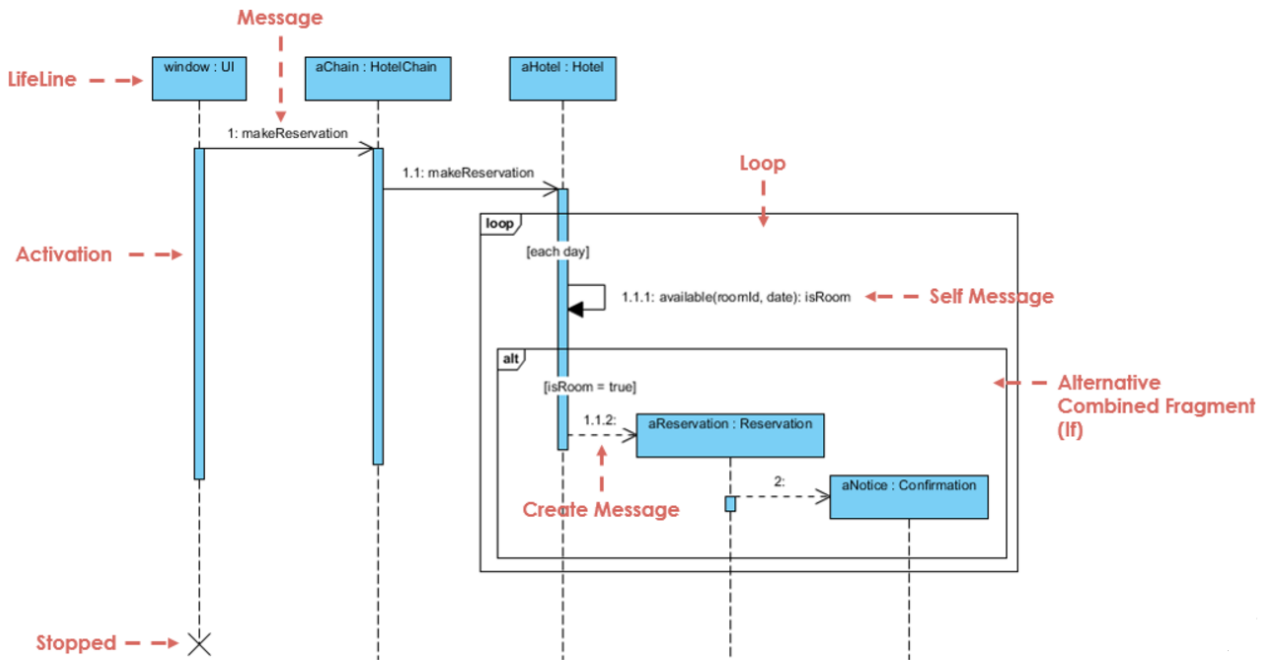
- **alt:** jedine ak je splnená podmienka sa budú dané procesy vykonávať
- **par:** fragment beží paralelne
- **loop:** časť procesov sa vykonáva opakovane pokiaľ je podmienka splnená

Ako môžme vidieť sekvenčný diagram poskytuje veľa notácií vhodných práve pre softvérové inžinierstvo. Robí to z neho ideálneho kandidáta na zobrazovanie a monitorovanie procesu v tele metódy. V dnešnej dobe už existujú nástroje, ktoré na tieto účely sekvenčný diagram využívajú. Sú nimi napríklad Enterprise Architect alebo Visual Paradigm.[3]

2.9 Diagram aktivít

Diagram aktivít je behaviorálnym typom, ktorý opisuje dynamické aspekty systému. Môžeme ho identifikovať aj ako pokročilejšiu formu flowchartu, ktorou vieme modelovať tok z jednej aktivity do druhej. Vďaka modelovaniu vieme dobre popísať koordináciu aktivít, ktoré vedú napríklad k správnemu poskytovaniu služby. Samozrejme, počas vykonávania aktivít môžu nastať špecifické prípady. Na toto diagram aktivít myslí a poskytuje vhodne vizuálne prvky pre takéto prípady. Spravidla sa diagrám skladá z vrcholov. Sú nimi:

- **počiatočný uzol:** označuje začiatok vykonávania množiny akcií
- **akcia:** úloha, ktorá musí byť vykonaná
- **ukončovací uzol:** ukončuje celý proces aktivít ktoré sa vykonávali



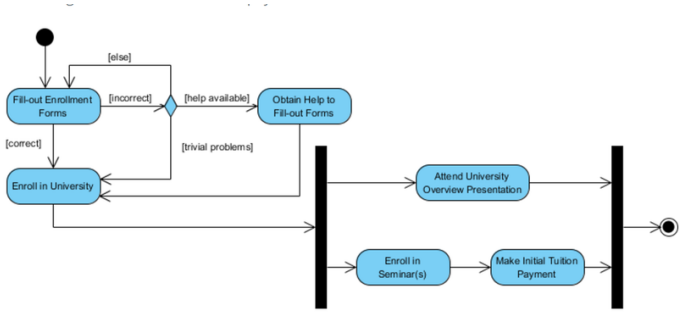
Obr. 2.6: Sekvenčný diagram
Zdroj:
https://www.cybermedian.com/id/a-comprehensive-guide-to-14-types-of-uml-diagram/

- **objektový uzol:** reprezentuje objekt, ktorý je prepojený s inými objektami
- **rozhodovací uzol:** prezentuje podmienku, ktorá zaisťuje že sa bude ďalej vykonávať práve jedna akcia
- **spájací uzol:** spája cesty rôznzch akcií do jedného uzlu, aby sa následne mohla vykonať ďalšia akcia.
- **uzol rozdelenia:** rozdelí tok na viacero paralelných akcií, ktoré sa v rovnakej fáze dejú naraz
- **uzol zjednotenia:** spája viaceré paralelné procesy do jedného uzla

V diagrame aktivít máme základný typ vzťahu a to tok objektov, ktorý znázorňuje tok či prechod z jednej akcie na druhú. Okrem toho máme nad týmto špecifikovaný objekt aktivity, ktorý nám práve zastrešuje množinu akcií, ktoré sa pri danej aktivite vykonávajú. Diagram nám tak ponúka ideálny spôsob ako vizualizovat telo metódy nejakého objektovo orientovaného kódu.[2]

2.10 UML-based Live Programming

Jedným z prístupov ako vyvíjať kód je vizualizovať ho na plátne. Je prirodzene atraktívnejšie písať riadky kódu do grafických objektov ako len do obyčajného textového



Obr. 2.7: Diagram aktivít
Zdroj: https://www.cybermedian.com/activity-diagram-a-quick-overview/

editora. V minulosti vznikli práve takéto prístupy, ktoré sa snažili kód vizualizovať na plátne. Jedným z najznámejších bol práve CodeBubles. Tento prístup sa ním inšpiroval a skúmal metódu vývoja softvéru vo virtuálnej realite pomocou kombinácie sekvenčných diagramov a diagramu tried.[9]

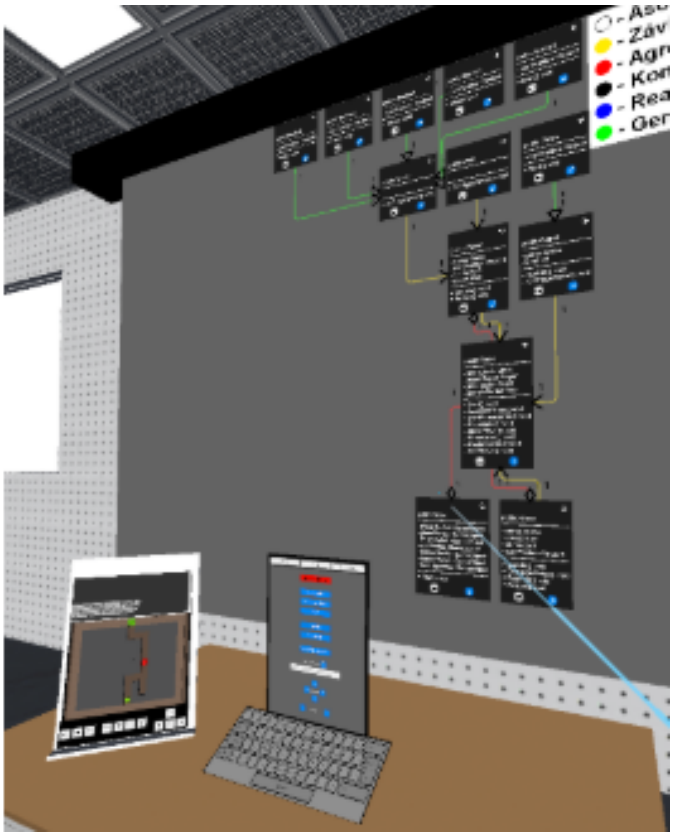
Sekvenčné diagramy reprezentovali kód tela jednotlivých metód. Diagramy tried zase štruktúru triedy, jej premenné a nazvy metód. Celé programovanie prebiehalo naživo. Ak užívateľ vykonal zmenu v kóde premietla sa hneď v diagrame a naspäť. Kompiláciu medzi kódom a diagramom zabezpečovala dobrá vnútorná reprezentácia kódu. Tá pozostávala z AST stromovej reprezentácie kódu a následne upraveného JSON súboru. Aplikácia bola vyvíjaná v hernom nástroji Unity, ktorý umožňuje vyvíjať aplikácie pre virtuálnu alebo aj zmiešanú realitu. Na správne rozmiestnenie objektov sa používala knižnica MSAGL. Kód bol zadávaný pomocou virtuálnej klávesnica alebo hlasom.[9]

Tento prístup priniesol inováciu a atraktívnosť do oblasti živého programovania pomocou doteraz nie veľmi prebádaných UML diagramov, ale poukázal aj na nedostatky spojené so zadávaním kódu a efektivitou programovania.

2.11 Toward a VR-Native Live Programming Environment

V predošlej časti sme si predstavili programovanie pomocou sekvenčných diagramov a diagramu tried. Tento výskum priniesol do živého programovania interatkívny spôsob vizualizácie kódu a pootvoril priestor pre hľadanie nových prístupov k zadávaniu kódu.[7]

Rohit Mehra vytvoril aplikáciu vo virtuálnej realite, ktorá vizualizovala kód do objektov. Tie boli vykreslené následne na 2D plátne. Užívateľ mohol kedykoľvek daný objekt premiestniť a organizovať si tak plátno. Konverziu z bloku do kódu nám tu opäť zabezpečovala AST stromová reprezentácia kódu. Výsledok skompilovaného kódu sme následne mohli vidieť a takisto sa nám zobrazili aj chyby ak sa vyskytli v kóde. Celá



Obr. 2.8: Programovanie vo virtuálnej realite pomocou sekvenčného a diagramu tried [9]

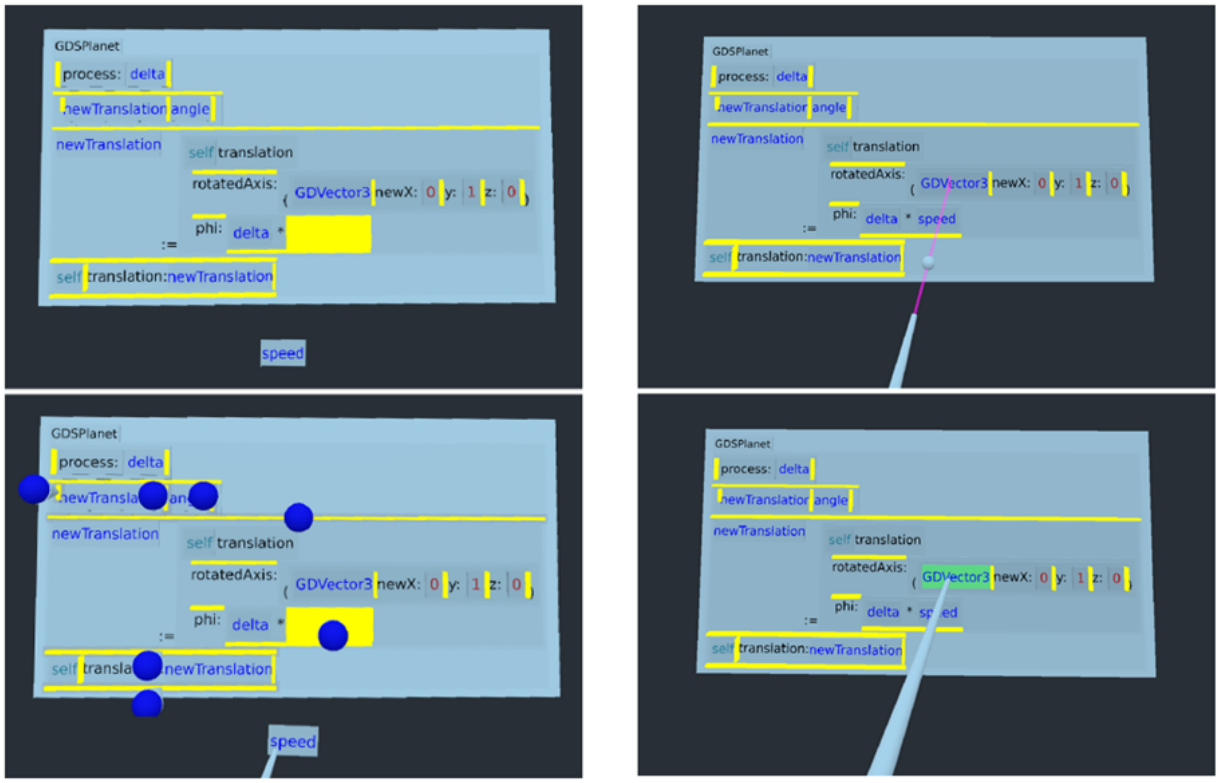
aplikácia bežala so systémom SmallTalk, ktorý bol pre živé programovanie maximálne vhodný. Aplikácia bola vyvinutá v hernom nástroji Godot, ktorý slúži na podobné účely ako Unity.[7]

Zadávanie kódu bolo riešené nasledovne. Virtuálne ruky nahradil prútik, ktorý slúžil ako ukazovátko. Pri nadídeni nad blok sa zobrazil vždý krátky vysvetľujúci popisok. Zadávanie kódu bolo umožnené pomocou kreslenia textu alebo Google klávesnicou. [7]

2.12 Generating Java Code from UML Class and Sequence Diagrams

Vývoj softvéru vo veľkých projektoch je finančne aj časovo náročný, preto sa firmy snažia vývoj softvéru urýchliť, a to napríklad inžinierstvom riadeným modelmi. Vďaka postupnej transformácii modelov sme schopní z nich získať implementáciu softvéru. Na takúto tvorbu nám môžu slúžiť aj UML modely.[11]

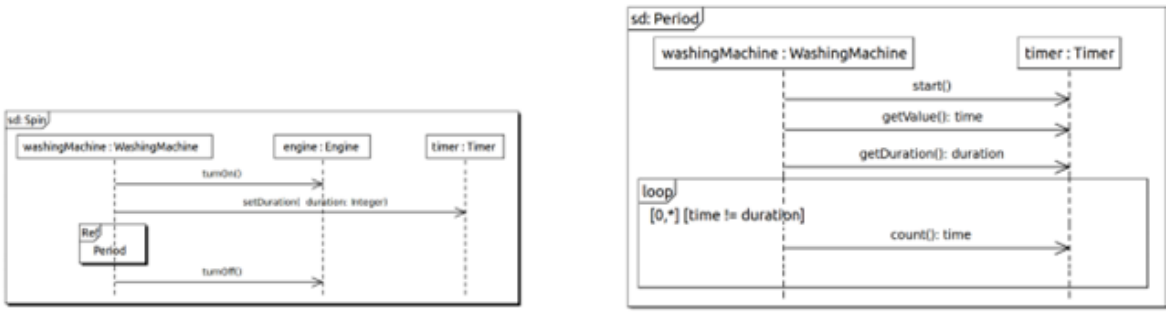
Výskum sa zameriava na generovanie kódu písaného v jazyku Java. Používa na to kombináciu diagramu tried a sekvenčných diagramov. V diagrame tried vieme efektívne znázorniť štruktúru triedy, jej dedičnosť a poprípade aj iné vlastnosti, ako sú



Obr. 2.9: Vizualizácia pomocou 2D objektov na plátne vo virtuálnej realite [7]

abstraktnosť a viditeľnosť voči zvyšku softvéru. V sekvenčnom diagrame sa zobrazujú jednotlivé volania, ktoré sa v metóde dejú. Takisto vieme z nich vytiahnuť aj informáciu o podmienkach či cykloch. Jednoduché aritmetické operácie v sekvenčnom diagrame nenájdeme.[11]

Pri transformácii UML do Java kódu uloží každú triedu diagramu tried do samostatného súboru. Pre skompilovateľnosť kódu musí mať aspoň jedna z tried zadefinovanú main metódu, z ktorej sa volajú následne ďalšie triedy. Tento fakt je v sekvenčnom diagrame poistený, keď trieda obsahujúca takúto metódu je prelinkovaná na hlavný sekvenčný diagram.[11]



Obr. 2.10: Ukážka sekvenčného diagramu znázorňujúceho volania metód [11]

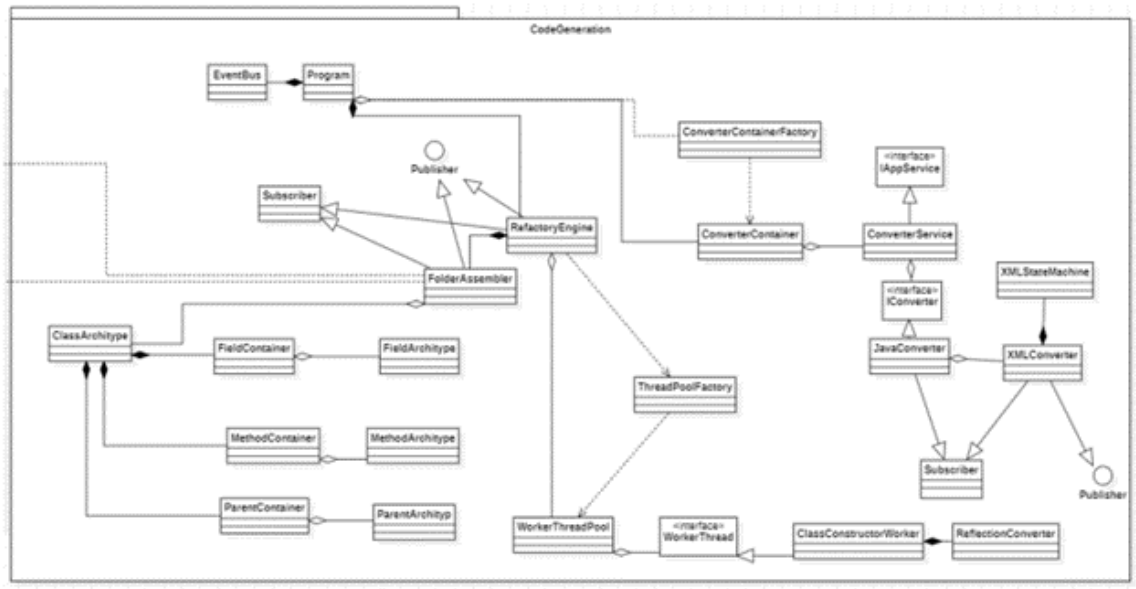
2.13 UML to code, and code to UML

V dnešnej dobe exsitujú rôzne prístupy pre generovanie kódu z UML diagramu. Môžme spomenúť aplikácie Enterprise Architects, Visual Pardigm alebo aj Star UML či Software ideas. Cieľom tohto výskumu bolo poskytnúť kompletný balík konverzie medzi kódom a UML diagramami. Pri testovaní softvéru sa merala rýchlosť, sledovalo sa vyťaženie CPU a internej pamäte.[5]

Aplikácia sa skladá z 3 komponentov: používateľské rozhranie, prevodník a komponent sledovateľnosti. Komponenty majú nezávislú implmentáciu. Postup konverzie z kódu je nasledovný:

- UML prevedieme pomocou XML konvertora do XML mapy
- Vytvoria sa pracovné vlákna, ktoré tieto informície preberú a vytvoria na základe nich typový objekt
- následne si objekty preberú generátory, ktoré na základe nich vygenerujú kód. Pre každý typ kódu sa používa iný typ generátoru

Ak došlo k nejakej zmene v kóde file listener túto zmenu zachytil a poslal do Code tracera, ktorý rozhodol, či zmena ovplyvní diagram UML. Na záver tejto kapitoly treba spomenúť modely umelej inteligencie, ktorá výrazným spôsobom pomáhajú konverzii z kódu do UML a naspäť. Vďaka schopnosti hĺbkového učenia je schopná lepšie analzyovať zdrojové kódy, ktoré vie spojiť s modelom a neskôr vytvoriť celú infraštruktúru založenú na algoritme učenia.[5]



Obr. 2.11: Proces generovania kódu [5]

Kapitola 3

Návrh

Počas analýzy vývoja softvéru sme identifikovali viaceré výzvy, ktoré môžu ovplyvniť jeho efektívnosť a prehľadnosť. Pri práci na rozsiahlych projektoch sa často stretávame s problémami, ako je neprehľadnosť a zložitosť implementácie, čo môže viesť k chybám a náročnej údržbe kódu. Tieto výzvy sa prejavujú najmä pri vizualizácii štruktúr a procesov v softvéri, kde je dôležité nájsť rovnováhu medzi detailnosťou a jednoduchosťou zobrazenia.

Ďalšou komplikáciou je interakcia používateľa s prostredím, najmä pri zadávaní a spracovaní kódu. Tradičné metódy môžu byť limitované, čo sťažuje intuitívnu a efektívnu prácu. Preto je nevyhnutné hľadať riešenia, ktoré zlepšujú vizualizáciu a zjednodušujú prácu s kódom, čím sa zvyšuje produktivita a kvalita výsledného softvéru.

3.1 Definícia a stanovenie cieľov

Cieľom vyvíjaného softvéru je umožniť programovanie v prostredí virtuálnej reality (VR) pomocou vizuálnych nástrojov, ako sú activity diagramy a triedy (class). Používateľ bude môcť vizualizovať a upravovať UML diagramy a využívať interaktívne funkcie bez potreby použitia tradičnej klávesnice, čo zjednoduší a spríjemní proces práce. Tento prístup podporí intuitívnu interakciu a uľahčí pochopenie a manipuláciu so štruktúrou a tokmi programu.

Aplikácia bude schopná prevádzať UML diagramy do C kódu s možnosťou živého prekladu a zobrazenia skompilovaného kódu v reálnom čase. Používatelia budú mať možnosť sledovať priebeh kompilácie a ľahko diagnostikovať prípadné chyby, čo prispeje k efektívnejšiemu vývoju a jednoduchšiemu ladeniu kódu. Tento prístup podporuje nielen efektivitu a produktivitu vývoja, ale aj lepšiu prístupnosť a používateľskú skúsenosť.

Dúfame, že po implementácii získame odpovede na tieto otázky:

1. Dokážeme nájsť vhodné špecifikácie UML diagramov pre úplné vygenerovanie

kódu?

2. Vyrovná sa programovanie s podporou VR tradičnému vývoju softvéru?

3.2 Návrh riešenia

Návrh riešenia je zameraný na vytvorenie interaktívneho softvéru pre programovanie v prostredí virtuálnej reality (VR). Na vykresľovanie a manipuláciu s UML diagramami, ako sú triedy (class) a activity diagramy, slúži hlavné platno aplikácie. Každý uzol triedy je vybavený interaktívnymi tlačidlami, ktoré umožňujú používateľovi pridávať a odstraňovať metódy, atribúty a hrany, čím sa zjednodušuje proces úpravy a rozširovania štruktúry diagramu. Po kliknutí na metódu sa na platne automaticky zobrazuje activity diagram zodpovedajúci jej kódu.

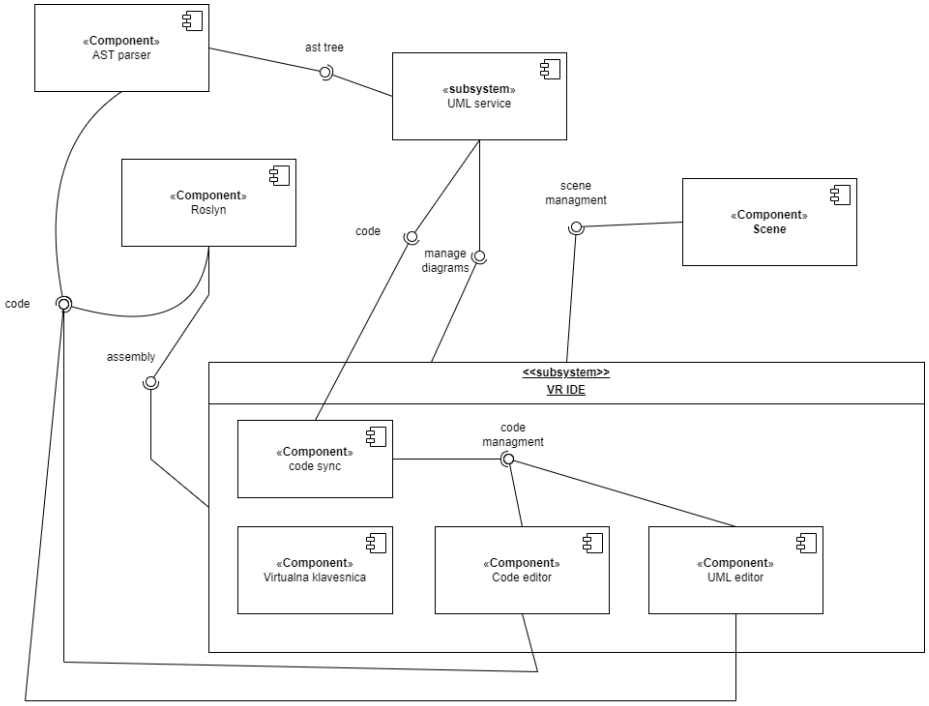
Vývoj softvéru bol realizovaný v prostredí Unity Engine, ktoré ponúka množstvo výhod v porovnaní s inými hernými enginmi, ako sú Godot či Unreal Engine. Unity je známe svojou rozsiahlou podporou pre vývoj VR aplikácií a poskytuje robustnú knižnicu nástrojov pre vývoj aplikácií s vysokou interaktivitou a vizuálnou kvalitou. Jeho flexibilita, široké komunitné zdroje a podpora pre rôzne platformy robia z Unity ideálnu voľbu pre tento typ projektu.

Pri načítaní a analýze kódu pre konverziu do UML diagramov sa využíva technológia C Roslyn, ktorá umožňuje efektívne spracovanie a transformáciu zdrojového kódu. Roslyn ponúka široké možnosti pre analýzu a manipuláciu s C kódom, čo zaručuje vysokú presnosť pri generovaní diagramov a transformáciách kódu. Jeho použitie prispieva k vyššej flexibilite pri práci s rôznymi kódovými štruktúrami a zjednodušuje integráciu s aplikáciou. Dôležitou súčasťou spracovania kódu je aj synchronizácia medzi UML diagramami a kódom, ktorá je založená na AST (Abstract Syntax Tree) reprezentácii kódu. Táto reprezentácia umožňuje efektívne sledovanie zmien v kóde a ich automatickú aktualizáciu v diagrame, čím sa zabezpečuje konzistentnosť a presnosť medzi vizuálnou a kódovou časťou aplikácie.

Na správne rozloženie elementov v diagramoch sa používa knižnica MSAGL, konkrétne Sugyama layout, ktorý poskytuje efektívne algoritmy na rozmiestnenie uzlov a hrán. Tento layout zjednodušuje čitateľnosť diagramov, zaisťuje logickú a vizuálnu súvislosť medzi prvkami a minimalizuje prekrývanie, čím sa zlepšuje celková používateľská skúsenosť.

Celý projekt beží na platforme MRTK (Mixed Reality Toolkit), ktorá ponúka stabilné a flexibilné prostredie pre vývoj aplikácií vo VR. MRTK umožňuje jednoduchú integráciu s Unity a poskytuje širokú škálu predpripravených interakčných prvkov a funkcionalít, ktoré zvyšujú efektivitu vývoja a poskytujú bohatý set nástrojov na tvorbu interaktívnych VR aplikácií.

Na kompiláciu a spracovanie kódu je taktiež využívaná technológia Roslyn, ktorá umožňuje živú kompiláciu a zobrazenie chýb v debug logu aplikácie. Týmto spôsobom je používateľ informovaný o chybách v reálnom čase, čo prispieva k rýchlejšiemu a efektívnejšiemu procesu vývoja a ladenia aplikácie.



Obr. 3.1: Komponent diagram navrhovaného systému

Kapitola 4

Implementácia

Prvotná fáza vývoja sa zameriavala na vykresľovanie a manipuláciu s UML diagramami v prostredí Unity. Prioritou bolo správne zobrazenie vzťahov medzi triedami v class diagrame, vrátane dôležitých prvkov ako generalizácia, kompozícia a ďalších typov asociácií, pričom sa dbalo na to, aby diagram nebol preťažený nadbytočnými informáciami. Dôležité bolo aj efektívne extrahovanie kľúčových prvkov z kódu, pričom sa zohľadnili názvy tried, atribúty a metódy. Pri ich zobrazení boli zahrnuté atribúty viditeľnosť, typ, návratová hodnota či vstupné parametre. Analýzu a načítanie kódu zabezpečila technológia C Roslyn, ktorá poskytla potrebnú flexibilitu pri spracovaní a generovaní reprezentácie kódu.

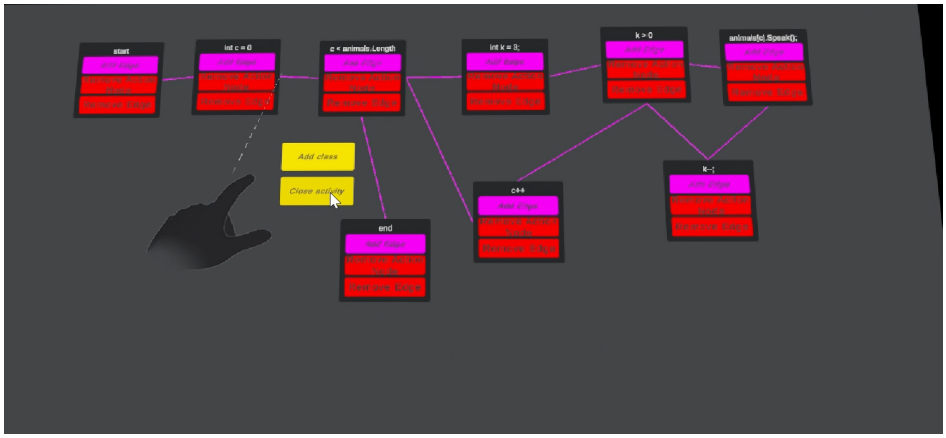
Pre podporu interaktívneho kreslenia diagramov boli objekty vybavené tlačidlami na pridávanie a odstraňovanie hrán, čo umožnilo dynamickú editáciu. Okrem toho boli objekty implementované ako pohyblivé s podporou funkcie drag-and-drop, čo umožnilo používateľovi intuitívne usporiadať prvky diagramu. Pri aktualizácii polohy objektov sa automaticky aktualizovali aj hrany, čím sa zabezpečila správna konektivita medzi prvkami. Na rozloženie diagramov bol použitý algoritmus MSAGL, ktorý zabezpečil hierarchické usporiadanie uzlov podľa dedičnosti tried, čím sa zvýšila prehľadnosť a čitateľnosť.

Okrem class diagramov bola vytvorená samostatná plocha na zobrazenie activity diagramov, medzi ktorými bolo možné ľahko prepínať pomocou funkcie SetActive. Activity diagramy boli navrhnuté tak, aby detailne reprezentovali logiku metód vrátane riadiacich štruktúr, ako sú cykly a podmienky. Pri implementácii sa objavili špecifické výzvy spojené so zobrazovaním riadiacich konštruktov, ako sú for, while a if-else, ktoré bolo potrebné vizualizovať jasne a intuitívne. Tieto konštrukty si vyžadovali špeciálny prístup k vykresľovaniu uzlov a hrán, aby bola logika programu čitateľná a ľahko sledovateľná.

Cyklus for a while boli reprezentované ako uzly obsahujúce podmienku spolu so štruktúrou slučky, pričom hrany určovali vstupné a výstupné cesty. Pre if-else kon-

štrukty boli zavedené rozvetvené uzly, ktoré obsahovali podmienku a dve vetvy – pravdivú a nepravdivú. Tento spôsob zobrazovania poskytol jasný vizuálny prehľad o logike programu, pričom sa kládol dôraz na zachovanie konzistencie a zrozumiteľnosti aj pri zložitejších štruktúrach.

Týmto prístupom sa dosiahlo prepojenie medzi logickou štruktúrou kódu a jeho vizualizáciou, čo uľahčuje pochopenie a analýzu programového toku. Používateľ tak má možnosť nielen sledovať priebeh algoritmu, ale aj rýchlo identifikovať chyby a slabé miesta v implementácii.



Obr. 4.1: Diagram aktivít prvotná implementácia

Kapitola 5

Výskum

Cielom nášho výskumu je zistiť programovanie pomocou diagramu tried a aktivít môže znížiť pracovné zaťaženie vývojarov a vyrovnať sa tak dostupným technológiam. Na tieto otázky vieme odpovedať časom dokončenia úloh a množstvom chýb, ktoré sa pri výskume udiali.

Pri evaluácii sa zameriavame na porovnanie VR prototypu, kde vieme programovať pomocou UML voči tradičným prístupom. Pripravili sme si jednoduché úlohy, zamerané na tvorbu programu pomocou UML diagramov pre architektúru a samotnú implementáciu pomocou zdrojového kódu.

5.1 Testovacie scenáre

Zámerom testovania je overiť používanie UML a klasickej techniky programovania. Po vyhodnotení chcem zistiť, či má potenciál vývoj softvéru cez diagram aktivít miesto sekvenčného diagramu. Úlohy boli vypracované pomocou VR IDE ale aj v Unity3D

1. Oboznámenie respondenta s prostredím
   (a) Respondent dostane na začiatku materiály, kde sa oboznámi s používaním prostredia
   (b) Respondent si následne vyskúša prácu s prostredím na nejakej jednoduchom programe
   (c) Zaznamenávanie si prvých skúsenosti respondenta s UML a Unity
2. Náčítanie grafu - platí iba pre VR prostredie
   (a) V prototype načítame predpripravený graf pomocou poskytnutej funkcionality
   (b) Načítaný graf usporiadame

3. Vykonanie prvej úlohy zameranej na doplnenie funkcionality
    (a) pomocou ponúkanej funkcionality, respondent doplní alebo vymaže jednotlivé elementy v UML diagrane
    (b) respondent doplnenú funkcionalitu otestuje pomocou tlačidla na generovanie kódu
    (c) zaznačíme si čas splnenia úloh, vzniknuté problémy prip plnení úlohy
4. Vykonanie druhej úlohy zameranej na porozumenie programu a následne zodpovedanie otázok týkajúcich sa fungovania programu
    (a) pomocou ponúkanej funkcionality, respondent získa informácie o programe
    (b) pomocou krátkeho dotazníka respondent odpovedá na konkrétne otázky
    (c) výsledky dotazníku vyhodnotíme
5. Vykonanie tretej úlohy zameranej na identifikovanie chyb programu a ich následne odstránenie
    (a) pomocou ponúkanej funkcionality, respondent nájde chyby v programe a ich opraví
    (b) zaznačíme si čas splnenia úloh, vzniknuté problémy prip plnení úlohy
    (c)
6. uloženie práce
    (a) upravený graf uložíme
7. vyplnenie UEQ dotazníku
8. vyplnenie doplňujúcich otázok

5.2 Stratégia evaluácie

Evaluácia navrhovaného riešenia sa zameriava na posúdenie jeho použiteľnosti a efektivity v porovnaní s tradičnými metódami vývoja softvéru. Hlavným cieľom je potvrdiť, či VR prostredie prináša pridanú hodnotu pri práci s UML diagramami, a zistiť, do akej miery je možné minimalizovať potrebu manuálneho písania kódu. Testovanie prebieha v dvoch režimoch – s použitím VR headsetu a s využitím emulátora, čo umožňuje priame porovnanie oboch prístupov.

Jedným z hlavných parametrov evaluácie je čas potrebný na vykonanie jednotlivých úloh. Tento kvantitatívny ukazovateľ poskytuje presný základ na porovnanie efektivity

práce vo VR a na emulátore. Meranie času je dôležité nielen pre objektívne porovnanie dvoch prístupov, ale aj na identifikáciu problémov. Výrazne dlhší čas na splnenie úlohy môže indikovať zložitosť interakcie alebo neintuitívne ovládanie. Takéto zistenia následne poskytujú priestor na optimalizáciu návrhu systému a skracovanie času potrebného na splnenie úloh.

Používateľská skúsenosť bude hodnotená pomocou dotazníka User Experience Questionnaire (UEQ). Tento nástroj je vhodný na meranie subjektívnych aspektov interakcie, ktoré nie sú zachytené v kvantitatívnych časových meraniach. UEQ umožňuje hodnotiť faktory, ako sú jednoduchosť použitia, efektivita, atraktivita a spokojnosť používateľov. Získané výsledky poskytujú cenné údaje na porovnanie VR prístupu s tradičnými metódami programovania. Navyše, nízke hodnotenie niektorých faktorov môže slúžiť ako podnet na ďalšie zlepšenia používateľského zážitku.

Počas testovania sa budú systematicky dokumentovať problémy, s ktorými sa účastníci stretnú. Táto dokumentácia umožní hĺbkovú analýzu príčin zlyhaní, identifikáciu možných prekážok v interakcii a poskytne podklady na iteratívne vylepšovanie systému. Zároveň tieto dáta pomáhajú validovať výsledky a vysvetliť prípadné rozdiely medzi prístupmi.

Okrem toho budú účastníkom kladené doplnkové otázky zamerané na ich subjektívne vnímanie práce vo VR v porovnaní s tradičným prístupom. Zamerajú sa na pohodu pri používaní VR, jeho dopad na produktivitu a identifikáciu oblastí, kde by bolo možné systém vylepšiť. Tieto otázky doplnia kvantitatívne a kvalitatívne dáta a zabezpečia komplexný obraz evaluácie.

Literatúra

[1] UML Class Diagram Tutorial — visual-paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/. [Accessed 05-12-2024].

[2] What is Activity Diagram? — visual-paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/. [Accessed 05-12-2024].

[3] What is Sequence Diagram? — visual-paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/. [Accessed 05-12-2024].

[4] Fahad Alhumaidan. A critical analysis and treatment of important uml diagrams enhancing modeling power. *Intelligent Information Management*, 04, 01 2012.

[5] Dumitru-Cristian Apostol, Paul-Daniel Rusovan, and Marius Marcu. Uml to code, and code to uml, a view inside implementation challenges and cost. pages 140–145, 10 2022.

[6] Creatly. Uml diagram types guide: Learn about all types of uml diagrams with examples. https://creately.com/blog/diagrams/uml-diagram-types-examples/, Accessed: 28.9.2022.

[7] Leonard Geier, Clemens Tiedt, Tom Beckmann, Marcel Taeumel, and Robert Hirschfeld. Toward a vr-native live programming environment. pages 26–34, 11 2022.

[8] Hacom. Kedy vznikol prvý počítač? objavte históriu počítača. https://www.hacom.sk/blog/category/blog/article/kedy-vznikol-prvy-pocitac-objavte-historiu-pocitaca.xhtml, Accessed: 21.07.2023.

[9] Jakub Kucecka, Juraj Vincur, Peter Kapec, and Pavel Cicak. Uml-based live programming environment in virtual reality. pages 177–181, 10 2022.

[10] Mino. Elektronick&xE1; u&x10D;ebnica pedagogick&xE9;ho v&xFD;skumu — e-metodologia.fedu.uniba.sk. http://www.e-metodologia.fedu.uniba.sk/index.php/kapitoly/veda-a-vyskum/vyskum-a-objektivna-realita.php?id=i1p2. [Accessed 09-12-2024].

[11] Abilio Parada, Eliane Siegert, and Lisane Brisolara. Generating java code from uml class and sequence diagrams. pages 99 – 101, 12 2011.

[12] TECHBOX. V kóde je dnes všetko! spoznajte históriu programovacích jazykov. https://www.techbox.sk/v-kode-je-dnes-vsetko-spoznajte-historiu-programovacich-jazykov, Accessed: 22.6.2022.

[13] Yashwant Waykar. Significance of class diagram in software development. 02 2014.

Zoznam obrázkov

Zoznam tabuliek